Searching Local Information in Mobile Databases [*]

Ouri Wolfson Bo Xu Huabei Yin Hu Cao
University of Illinois at Chicago
{wolfson, boxu, hyin, hcao2}@cs.uic.edu

1. Introduction

A mobile ad-hoc network (MANET) is a set of moving objects that communicate with each other via unregulated, short-range wireless technologies such as IEEE 802.11, Bluetooth, or Ultra Wide Band (UWB). No fixed infrastructure is assumed or relied upon. An important application domain of MANET's is local resource discovery. In a local resource discovery application, a user finds local resources that satisfy specified criteria. For example, a driver finds an available parking slot in a region by receiving information generated by the parking meter, or gets the traffic conditions on a highway segment a mile ahead; a cab driver finds a near-by customer, or a participant at a convention finds another participant with a matching profile.

The main problem in answering queries about resources in MANET's is finding the resource information, since the issuer of a query usually does not know the network-id of the moving objects storing the resource information, or how to reach them. In this paper we propose an algorithm for mobile resource discovery in MANET's in which the moving objects disseminate *resource-information* (namely reports) and *queries*. Each moving object can be a report producer, consumer (i.e. query issuer), or both. The MANET is self organizing in the sense that the reports flow towards the appropriate queries, even while the network topology changes.

Resource-reports and queries correspond to resource descriptions and resource queries in the resource discovery literature, and events and subscriptions in the publish/subscribe literature. Existing literature on resource discovery in MANET's addresses this problem using a *pull* approach. Specifically, a moving object floods a resource discovery query in the MANET, and when found the network constructs a routing structure from the query originator to the resource. The structure is built by augmenting traditional MANET routing protocols. Existing literature on publish/subscribe in MANET's also has queries (called profiles) and resources (called events), and the objective is to route the events to the matching subscribers. Again, a routing structure is built and maintained, but the events are *push*ed along the structure to the appropriate profiles, as they are produced. In short, both push and pull approaches use routing structures, and consequently they can be inefficient, particularly in high mobility networks which are prone to frequent topology changes. In such an environment, either a lot of communication has to be expended to keep the routing structure up to date, or the routing structure rapidly becomes obsolete and misses many matches.

In this paper we propose a hybrid approach between push and pull, in the sense that it disseminates both queries and resource reports. In this approach, a moving object O periodically selects the k most relevant reports and queries in its local database and broadcasts them to its neighbors (i.e. the objects that are within the transmission range of O). Upon receiving the broadcast from O, each neighboring object incorporates the received reports and queries into its own local database, and subsequently broadcasts the top k reports and queries. Thus reports and queries transitively spread across moving objects. The frequency of broadcasting and the size of each broadcast (i.e. k) depend on the bandwidth and power allocated by a moving object to local resource discovery. We call this algorithm *Periodic Flooding with Reports Ranking (PF_Rank*).

A novel aspect of the PF_Rank algorithm is the strategy used by a moving object O to rank the reports based on their relevance. Intuitively, the relevance of a report depends on the queries in the local database (which represent the global demand in the network); the more queries it satisfies, the more relevant is the report.

While disseminating queries enables moving objects to rank reports based on the global demand, it reduces the bandwidth and power available for dissemination of reports. Thus there is a tradeoff when dividing the space (k) of a broadcast between queries and reports. In this paper we experimentally determine the optimal fraction of queries in a broadcast message.

In order to evaluate the proposed PF_Rank algorithm, we experimentally compare it with periodic flooding without ranking (referred to as *PF_Random*), in which no queries are disseminated and the k reports are randomly selected. We compare these two algorithms in terms of the average number of relevant reports that are delivered to a consumer. We compare the algorithms in low mobility (simulating a pedestrian application); and for 802.11g. The experimental results show that PF_Rank

[*] Research supported by NSF Grants 0326284, 0330342, ITR-0086144, 0513736, and 0209190.

performs better than PF_Random. The benefit of report-ranking is greater when the bandwidth and power allocation to local resource discovery is small, and when the demand pattern is skewed, i.e., a few resources are very popular and many resources are unpopular.

2. The Model and PF_Rank Algorithm

Our system consists of a finite set of point (i.e. without an extent) *moving objects*. Their motion occurs in a finite geographic space. Occasionally, a moving object O obtains from an outside source information about a *resource*; the resource has some unique *resource-id*. For example, a pedestrian passing by an Automatic Teller Machine learns about the location and Bank of the machine from a Wi-Fi transmitter in the machine. The information about a learnt resource R is represented by a *report*, denoted $a(R)$; O is called the *producer* of that report.

Each moving object O also issues *queries* that express O's interests in certain types of information. O is called the *issuer* of these queries. An example of a query is a request for the locations of ATM's within 4 blocks of the Sears Tower in Chicago. The queries issued by a moving object O are *native* to O. For simplicity we assume that at any point in time a moving object has a single native query. A report $a(R)$ either satisfies a query Q, or not. If it does then $Q(a(R))$ = *true* and *match*$(a(R), Q) = 1$. Otherwise $Q(a(R))$ = *false* and *match*$(a(R), Q) = 0$.

Now, let us describe our PF_Rank algorithm. The objective of PF_Rank is to maximize the number of reports delivered to the "right" moving objects, i.e., to the issuers of queries satisfied by these reports. For this purpose, both queries and resources reports are disseminated. The purpose of disseminating the queries is to reflect the demand in the neighborhood and the network in general, such that the reports that are in demand are ranked higher. In other words, at any point in time a moving object's database has reports, its native query, and queries issued by other moving objects.

The PF_Rank algorithm works as follows. Every p seconds, a moving object O sorts reports in its database based on a ranking function, and selects the $k(1\text{-}q)$ highest ranked reports and $k \cdot q$ queries to broadcast. The $k \cdot q$ queries include the native query of O and the rest are randomly selected. k, p, q are the *broadcast size*, *broadcast period*, and *query fraction* respectively. The values of k and p are determined by the cost a moving object is willing to pay in terms of bandwidth and power for local resource discovery applications. The appropriate value of q is determined experimentally in this paper.

The rank of a report is defined to be the sum of all its relevance values. Let the *foreign* queries in moving object O's queries relation be queries that are received from other objects, i.e. non-native queries.

Definition 1: Let $Q_1, Q_2, \ldots, Q_n$ be the foreign queries in moving object O's queries relation. The *rank of a report* $a(R)$ *at* O *at time* t is:

$$rank(a(R)) = \sum_{i=1}^{n} Match(a(R), Q_i) \qquad (1)$$

Notice that the native queries do not participate in computation of report rank, since such a query should not increase the probability of a report being propagated. Furthermore, for simplicity of presentation we assume that the size of a query is equal to the size of a report (although our results easily extend to the different-size case) and that an infinite number of queries and reports fit in the memory of a moving object (to avoid dealing with memory allocation issues).

3. Experimental Analysis

Simulation Method. All algorithms are implemented in SWANS, a Scalable Wireless Ad-hoc Network Simulator built by Cornell University. In the simulations, we use 802.11g with the data transmission speed of 54M bits per seconds. 100 objects move within a 0.5mile×0.5mile square area according to the following *random way-point* mobility model. A moving object is introduced at a random point in the area, and it moves at constant speed in a straight line to another random point. There it pauses, then moves at constant speed in a straight line to another random point; and so on. The pause time at each waypoint is a random variable uniformly distributed between 120 and 240 seconds. The motion speed is randomly picked up from the interval [0.5, 1.5] miles/hour. This mobility model simulates, for example, the motion of a shopper at a mall. The whole simulation runs for 18000 seconds. Each object has a life span which is set to be a random period between 600 seconds and 1200 seconds. When the life span of an object expires, it is removed from the system and a new object is created. Thus the number of *live* moving objects is fixed.

For representing resources and queries, we adopted the Number Intervals (NI) subscription model introduced in [1]. Particularly, a resource is represented by a point within the real interval [0, 1]. A query is represented by a range within [0, 1], e.g., [0.2, 0.7]. A report $a(R)$ matches a query Q if R's number falls into Q's range.

One resource is generated every second. The number that represents the resource, called R, is randomly chosen from the [0, 1] interval. An arbitrary live moving object becomes the producer of the report $a(R)$.

Each moving object has a native query which is generated when the object is introduced to the system, and is fixed for the life span of the object. The range of the query is generated by choosing a center and a length. The length of the range is selected randomly according to a normal distribution with mean 0.05 and variance 0.002.

The query-center falls into the [0, 1] interval following a Zipf distribution. In particular, the [0, 1] interval is divided into 10 disjoint sections ([0, 0.1), [0.1, 0.2), …). The probability that a query-center falls into the *i*-th ($1 \le i \le 10$) section is $(1/i)^s / \sum_{j=1}^{10} (1/j)^s$, where *s* is called the *relative popularity* and *i* is called the *popularity rank*. *s* ranges from 0 to 4. The greater the value of *s*, the more skewed is the distribution of queries. In other words, the resources are uniformly distributed, and the queries are distributed according to Zipf's law.

The query fraction *q* ranges from 0 to 0.9 with an increment of 0.1. The case in which the query fraction is 0 is the PF_Random algorithm. The broadcast period *p* ranges from 50 to 300 seconds; the broadcast size *k* is fixed to 10 reports. By varying the value of the broadcast period parameter, we model different amounts of bandwidth/power allocation to local resource discovery.

Performance Measure. As the performance measure we use the average number of matching resource reports received by a moving object during its trip. In other words, the performance of a dissemination algorithm is the average number of matches that the algorithm delivers to each moving object.

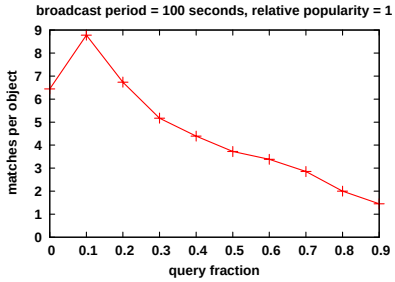

Figure 1. Tuning-up query fraction for PF_Rank (query fraction 0 corresponds to PF_Random)

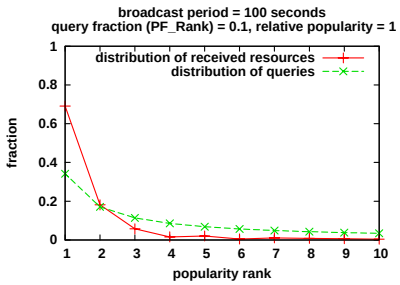

Figure 2. Distribution of received resources with PF_Rank versus distribution of queries

Simulation Results. We first tuned up PF_Rank by finding the query fraction *q* that generates the best performance. From Figure 1 can be seen that PF_Rank reaches the best performance when *q*=0.1, for the given parameter configuration. Actually, we varied the broadcast period, broadcast size, and skew, and in all cases the optimal query fraction in the range [0.1, 0.9] is still 0.1 .

We also traced the distribution of the resource-reports that are received by the moving objects (see Figure 2). That is, what is the fraction of the resources that fall into the *i*-th interval (i.e. popularity *i*) out of all the resources that are received by the moving objects in the whole system throughout the simulation. The result shows that the distribution of delivered resources is consistent with the demand pattern (the distribution of queries), even with a small value of query fraction (*q*=0.1). This verifies the benefit of query dissemination and reports ranking.

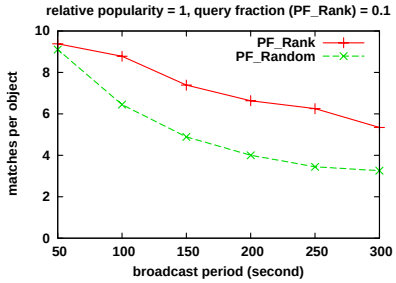

Figure 3. Performance versus broadcast period

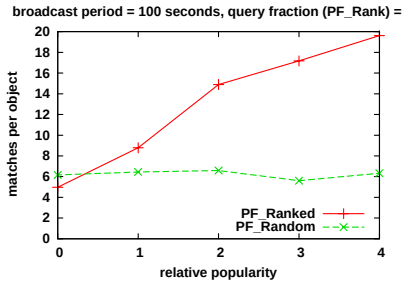

Figure 4. Performance versus relative popularity

Figure 3 shows the performance of PF_Rank and PF_Random as a function of the broadcast period. The advantage of PF_Rank over PF_Random (i.e. the ratio between the performance of PF_Rank and that of PF_Random) decreases as the broadcast period decreases. Intuitively, when the broadcast period is very small, PF_Random manages to deliver a lot of matching reports to moving objects by quick repeated transmissions; a report that is not selected in the last broadcast may be selected in one of the next few broadcasts which comes soon. This suggests that PF_Rank is particularly useful when the bandwidth/power allocation to local resource discovery is low. In other words, PF_Rank makes more bandwidth/power available to other network applications.

Figure 4 shows that the advantage of PF_Rank increases as the demand skew increases. This is intuitive, because random report dissemination is a good strategy if the demand is uniformly distributed.

[1] Y. Huang, H. Garcia-Molina: Publish/Subscribe in a Mobile Environment. *Wireless Networks*, 10(6): 643-652 (2004).